Product Specification

USB 3060
basic MOST 3060

MOST25 Interface

User Manual Version 1.3

GÖPEL electronic

Get the total Coverage!

GOEPEL electronic GmbH
Goeschwitzer Str. 58/60
D-07745 Jena
Phone: +49-3641-6896-597
Fax: +49-3641-6896-944
E-Mail: ats_support@goepel.com
http://www.goepel.com

printed: 08.11.2012

Issue: November 2012

1 Installation

1.1 Hardware Installation of USB 3060 Boards

Warning

Please make absolutely certain that all of the installation procedures described below are carried out with your system switched off.

For the hardware installation of a basic MOST 3060, only the cables for USB and Power supply (if applicable) have to be connected (see Hardware).

Warning

Electro Static Discharge (ESD) can harm your system and destroy electronic components. This can lead to irreparable damage on both the basic MOST 3060 or USB 3060 device and the belonging system as well as to unexpected malfunction of your test system.

Therefore do not touch the board surface or any connector pins and electronic components.

For the hardware installation of a USB 3060 board, open your USB system according to its conditions and select a free slot. Insert the board carefully into the prepared slot. Use the lever at the front plate in order to push in the board finally.

1.2 USB Driver Installation

For proper installation of the GOEPEL electronic USB driver on your system, we recommend to execute the G-USB driver setup.
To do that, start the *G-USB-Setup-*.exe* setup program (of the supplied CD, “*” stands for the version number) and follow the instructions.

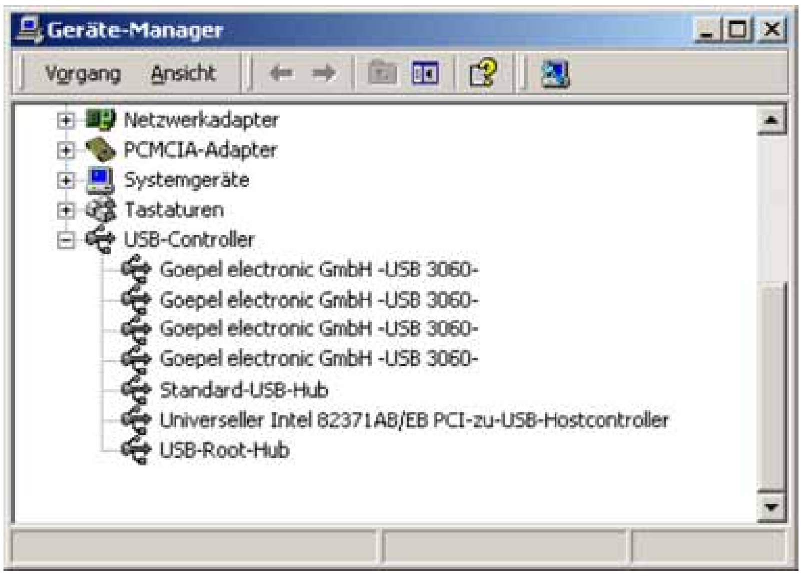
The available device driver supports Windows® 2000, 7 and XP systems.

The following step is only required in case you do not use the G-API.

If you want to create your own software for the boards, you possibly need additional files for user specific programming (**.LLB*, **.H*). These files are not automatically copied to the computer and have to be transferred individually from the supplied CD to your development directory.

After driver installation, you can check (for example by the Windows® Device Manager) whether the boards are properly embedded by the system.

Among others, the following figure shows the successful embedding of four USB 3060 boards or basic MOST 3060 devices (USB 3060):



Geräte-Manager

Vorgang Ansicht

- Netzwerkadapter
- PCMCIA-Adapter
- Systemgeräte
- Tastaturen
- USB-Controller
  - Goepel electronic GmbH -USB 3060-
  - Goepel electronic GmbH -USB 3060-
  - Goepel electronic GmbH -USB 3060-
  - Goepel electronic GmbH -USB 3060-
  - Standard-USB-Hub
  - Universeller Intel 82371AB/EB PCI-zu-USB-Hostcontroller
  - USB-Root-Hub

Figure 1-1: Display of Device Manager

Please note that the Device Manager shows ALL USB controllers supported by this *G-USB* driver.

2 Hardware

2.1 Definition

USB 3060 MOST25 boards are GOEPEL electronic communication boards with USB 2.0 interface (MOST = **M**edia **O**riented **S**ystems **T**ransport).

These boards are used in the field of media electronics, for example to test entertainment systems in automotive technology.

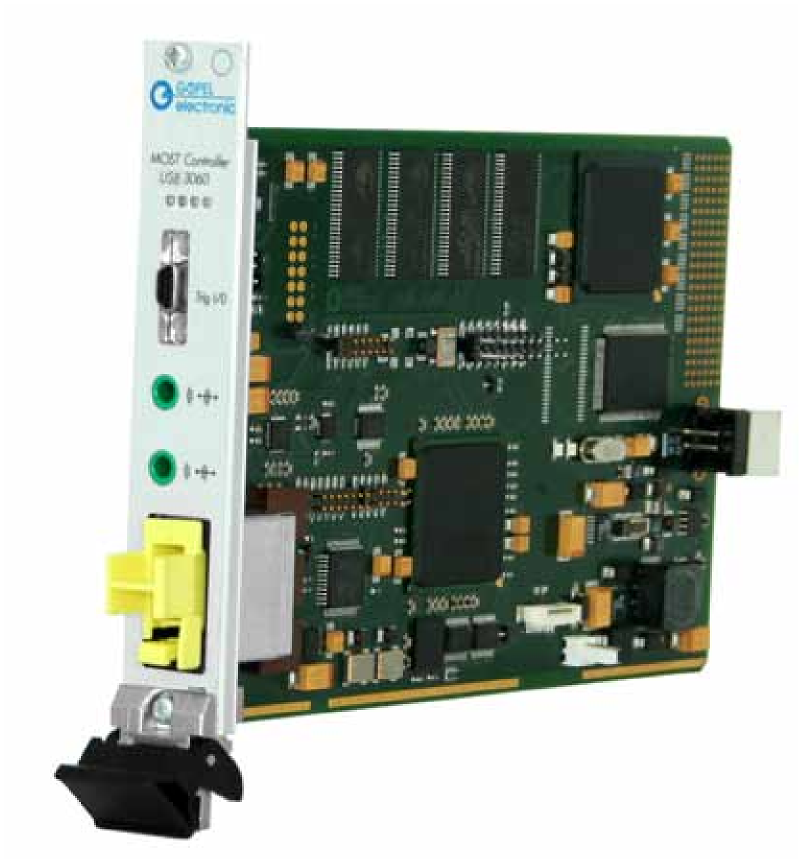


Figure 2-1: USB 3060

For operating USB 3060 boards you need the GOEPEL electronic USB Rack which can cover up to 16 GOEPEL electronic USB boards.

In this case, power supply comes from the built-in power supply unit.

USB 3060 boards offer the following resources:

- 1 optical MOST25 interface
- Support of three operation modes: Master, Slave, Bypass
- Spy Function: Possibility of monitoring the MOST data as passive bus partner (board in the Bypass mode) or as active bus partner (board in the Master or Slave mode)
- Independent timer onboard
  with a time stamp resolution of 8 ns
- Possibility of master frame rate change-over
  between 44.1 kHz and 48 KHz
- Analogue Audio LINE IN and LINE OUT connections
- Ring break diagnosis function via frontal plug connector
- Extended trigger functions with two trigger inputs and four trigger outputs to the frontal plug connector or to the backplane
- The MOST interface has a 32 bits µController
  (TriCore TC1796, 150MHz) with 8 Mbytes SRAM onboard
- Display of the controller states by four LEDs at the front panel (see LED Display)

basic MOST 3060 is a GOEPEL electronic stand-alone device based on a USB 3060 communication board to be connected to a PC or laptop.

It was in particular developed for applications out of complex test systems.

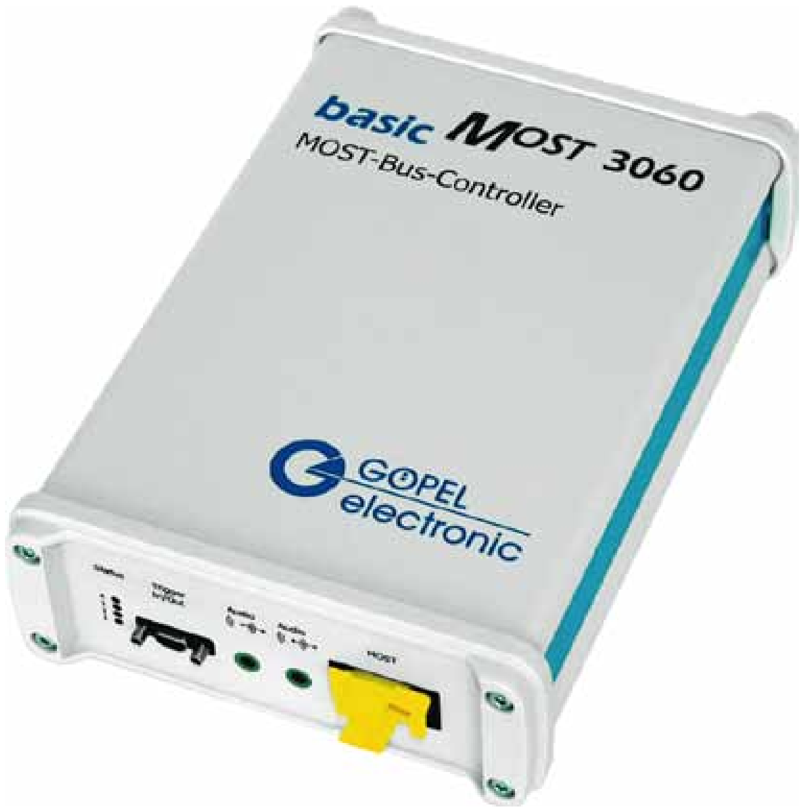


Figure 2-2:
basic MOST 3060

When operating basic MOST 3060 at passive USB hubs or laptops with low current capacity, power supply is effected via the two ext. Power Supply females (red = plus/ blue= minus) at the device's rear side (opposite to the MOST communication interface side).
The blue female is connected with the GND connections of the USB interface.

In the case your basic MOST 3060 is connected to an active USB Port with I_{Lmax} = 500 mA, power supply is also possible via this port.
Then, do not connect the ext. Power Supply females.

2.2 Technical Specification

2.2.1 Dimensions

(Width x Height x Depth):

- USB 3060: 4 HP x 130 mm x 185 mm
- basic MOST 3060: 130 mm x 55 mm x 200 mm

The dimensions stated for USB 3060 refer to a board inside the GOEPEL electronic USB Rack.

2.2.2 USB 3060 / basic MOST 3060 Properties

Symbol	Parameter	Min.	Typ.	Max.	Unit	Remarks
U_{BAT}	Battery voltage		12	27	V	Only relevant for Ring break diagnosis
U_{ext}	ext. Power Supply: External power supply for basic MOST 3060	8	12	25	V	In case of active USB Port with I_{Lmax} = 500 mA also possible via USB
	Transmission rate			25	MBaud	
	External trigger input	3,3		27	V	
	External trigger output		5,0		V	

Warning

Please use the delivered USB cable to connect USB 3060/ basic MOST 3060 devices to the PC’s USB interface.

Other cables may be inapplicable.

2.3 Construction

2.3.1 General

In their basic version, USB 3060 boards have one MOST25 interface. Figure 2-3 shows schematically the board construction as a block diagram.

For USB 3060/ basic MOST 3060 devices, the USB Controller provides the interface to the USB bus. It includes all the function blocks required for the communication with the USB bus.

The USB 3060 board, developed for the GOEPEL electronic USB Rack, has additional signal connections (GPIO) via the USB Connector. These connections can be interconnected for communication between several boards.
At basic MOST 3060 devices, GPIO connections are not used.

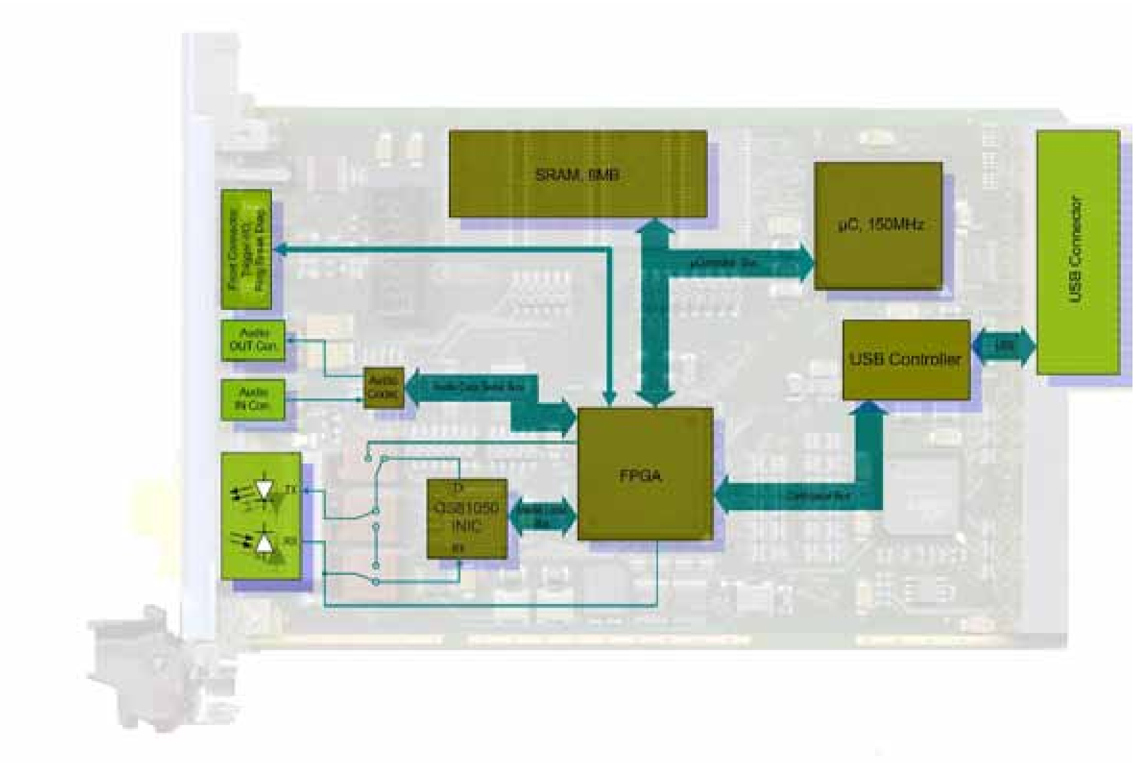


Figure 2-3: Block diagram of a USB 3060 Communication board

2.3.2 Addressing

Several USB 3060 boards in the GOEPEL electronic USB Rack are exclusively addressed according to their serial numbers
(see Software):
The board with the LEAST serial number is always the board with the number 1 (that means, DeviceNumber = 1.)

Tip

To improve clarity, we recommend to arrange several USB 3060 boards in the order of ascending serial numbers in the GOEPEL electronic USB Rack.

2.3.3 Trigger Behavior

The MOST interface provides two additional trigger input and four additional trigger output connections to the frontal plug connector to be interconnected by the corresponding driver configuration (not software supported for the time being).

2.3.4 MOST Interface

For proper operation of a MOST interface in a network, all partners must communicate with the same system clock. This depends, among others, on the Master frame rate of the system.

USB 3060 support the following Master frame rates:

- 48 kHz and
- 44.1 kHz

(G-API command G_Most_Node_SetProperties, parameter ClockMode).

2.3.5 Connector Assignment

Type: miniDSub 9 poles, male

The XS1 frontal connector offers four trigger outputs, two trigger inputs and the connections for the Ring break diagnosis interface to the user.

As an option, GOEPEL electronic can deliver a cable for connecting the frontal connector. One cable end is designed as a counterpart to XS1 (that means miniDSub 9 poles, female). The other end is not connected and can be configured by the user himself.

The connector assignment and the colors of the belonging wires are identical for USB 3060 and basic MOST 3060 according to the following table:

No.	XS1 contact	Signal name	Remarks	Wire color
1	1	TRGO_1	Trigger output 1	black
2	6	TRGO_2	Trigger output 2	green
3	2	TRGO_3	Trigger output 3	yellow
4	7	TRGO_4	Trigger output 4	blue
5	4	TRGI_1	Trigger input 1	orange
6	5	TRGI_2	Trigger input 2	red
7	3	GND	Ground potential	brown
8	8	U_{BAT}	Reference potential Ring break diagnosis	white
9	9	RingDiag	Data line Ring break diagnosis	magenta
			No function	gray

2.3.6 LED Display

The LEDs arranged at the front panel indicate the current operation state of the micro controller of the MOST interface.

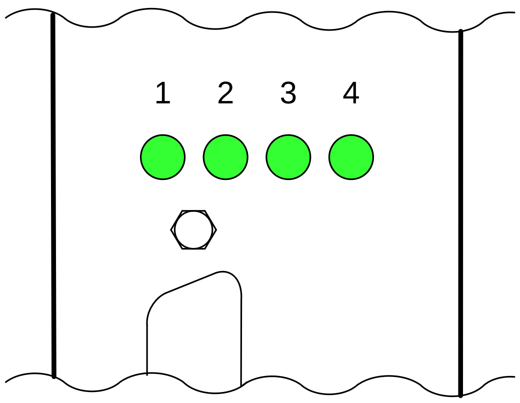


Figure 2-4
LED Display

The following table shows important display states of these LEDs:

State				**Remarks**
LED 1	**LED 2**	**LED 3**	**LED 4**	
Alternately blinking				Bootloader software runs; Error reason (probably): Software reset not executed
				Firmware runs
ON (shortly)				Display while Firmware commands are executed
		ON		MOST bus: Light on
			ON	MOST bus: Locked

This LED display is effected with low priority and can be affected by other running programs.

2.4 Delivery Notes

USB 3060 boards/ basic MOST 3060 devices are delivered as follows with one MOST25 interface per board/ device:

- USB 3060.00
- basic MOST 3060.00

As an option, the cable described in section Connector Assignment can be delivered:

- CAB 3060.10

3 Software

There are three ways to integrate USB 3060/ basic MOST 3060 hardware in your own applications:

- Programming via G-API
- Programming via DLL Functions
- Programming with LabVIEW

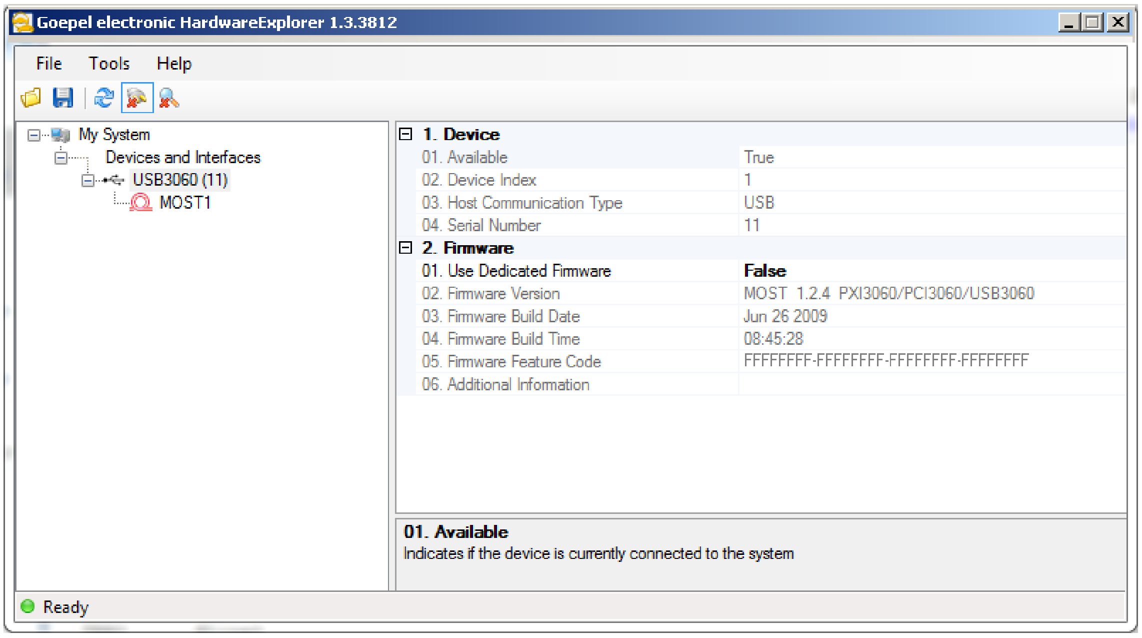
3.1 Programming via G-API

The G-API (GOEPEL-API) is the “C” user interface for GOEPEL electronic hardware under Windows®. So the G-API is the preferred programming environment for this hardware.
It provides a wide, hardware independent command set for CAN, LIN, K-Line, FlexRay, MOST, LVDS, ADIO and Diagnostic services.
No matter whether a PXI/ PCI, USB or Ethernet device is used – the commands remain the same.

The hardware abstraction introduced with the G-API gives the test application parallel access to the hardware, allowing one application to access multiple hardware interfaces, as well as multiple applications can access the same hardware interface in parallel.

Another feature introduced by the G-API is the asynchronous hardware access. This means no execution blocking for pending firmware commands. The command acknowledgement is provided via a callback mechanism.

With the HardwareExplorer GOEPEL electronic provides an efficient hardware configuration and management tool, offering users an easy way to manage their hardware configurations and identifying specific hardware interfaces by logical names. Using logical interface names in the application saves from rebuilding the application when porting it to another interface or controlling device, as the interface can be easily reassigned in the HardwareExplorer. Furthermore, the HardwareExplorer provides a simple means of testing the interaction between hardware and software by executing the integrated self-tests.



1. Device	
01. Available	True
02. Device Index	1
03. Host Communication Type	USB
04. Serial Number	11
2. Firmware	
01. Use Dedicated Firmware	**False**
02. Firmware Version	MOST 1.2.4 PXI3060/PCI3060/USB3060
03. Firmware Build Date	Jun 26 2009
04. Firmware Build Time	08:45:28
05. Firmware Feature Code	FFFFFFFF-FFFFFFFF-FFFFFFFF-FFFFFFFF
06. Additional Information	

01. Available
Indicates if the device is currently connected to the system

Figure 3-1: HardwareExplorer

Please consult the G-API documentation for further information. This documentation and the installation software are located in the *G-API* folder of the supplied “Product Information” CD.

3.2 Programming via DLL Functions

Programming via DLL Functions is possible also in future for existing projects which can not be processed with the GOEPEL electronic programming interface G-API.

We would be pleased to send the GOEPEL Firmware documentation to you on your request. Please get in touch with our sales department in case you need that.

The GUSB_Platform expression used in the following function description stands for the name of a GOEPEL electronic USB driver.

For the used structures, data types and error codes refer to the headers – you find the corresponding files on the supplied CD.

In this User Manual, Controller means ALWAYS the microcontroller assigned to the MOST interface of a USB 3060/ basic MOST 3060 device.

On the other hand, USB Controller means ALWAYS the controller providing the USB 2.0 interface of the USB 3060/ basic MOST 3060 device.

3.2.1 Windows Device Driver

The DLL functions for programming using the Windows device driver are described in the following sections:

- Driver_Info
- DLL_Info
- Write_FIFO
- Read_FIFO
- Read_ FIFO_Timeout
- Write_ COMMAND
- Read_ COMMAND
- Xilinx_ Version

3.2.1.1 *Driver_Info*

The GUSB_Platform_Driver_Info function is for the status query of the hardware driver and for the internal initialization of the required handles.

Executing this function at least once is obligatory before calling any other function of the GUSB_Platform driver.

Format:

```
int GUSB_Platform_Driver_Info(GUSB_Platform_DriverInfo *pDriverInfo,
                              unsigned int LengthInByte)
```

Parameters:

Pointer, for example pDriverInfo,
to a data structure
For the structure, see the *GUSB_Platform.h* file on the delivered CD

LengthInByte
Size of the storage area pDriverInfo is pointing to, in bytes

Description:

The GUSB_Platform_Driver_Info function returns information regarding the status of the hardware driver.

For this reason, the address of the pDriverInfo pointer has to be transferred to the function. By means of the LengthInByte parameter the function checks internally if the user memory is initialized correctly.

The function fills the structure pDriverInfo is pointing to with statements regarding the driver version, the number of all involved USB controllers (supported by this driver) and additional information, e.g. the serial number(s).

Making the hardware information available
as well as initializing the belonging handles is obligatory for the further use of the USB hardware.

3.2.1.2 DLL_Info

The GUSB_Platform_DLL_Info function is for the version number query of the DLL.

Format:

```
int GUSB_Platform_DLL_Info(GUSB_Platform_DLLinfo *DLLinformation)
```

Parameters

Pointer, for example DLLinformation,
to a data structure
For the structure, see the *GUSB_Platform.h* file on the delivered CD

Description:

The GUSB_Platform_DLL_Info function returns the DLLinfo structure. The first integer value contains the version number of the *GUSB_Platform.dll*.

Examples:

Version number **`1.23`** is returned as **`123`**,
and version number **`1.60`** as **`160`**.

3.2.1.3 *Write_FIFO*

With the GUSB_Platform_Write_FIFO function a command is sent to the Controller.

Format:

```
int GUSB_Platform_Write_FIFO(unsigned int DeviceName,
                             unsigned int DeviceNumber,
                             t_USB_FIFO_Interface_Buffer *pWrite,
                             unsigned int DataLength)
```

Parameters:

DeviceName

Type of the addressed device (number declared in *GUSB_Platform_def.h*, for USB 3060/ basic MOST 3060 = 18)

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has <u>always</u> the DeviceNumber 1).

Pointer, for example pWrite,
to the write data area

DataLength

Size of the storage area pWrite is pointing to, in bytes
Data is consisting of Command Header and Command Bytes
(currently max. 4096 bytes per command)

Description:

The GUSB_Platform_Write_FIFO function sends a command to the Controller.

For the general structure, see the General Firmware Notes section of the GOEPEL Firmware document.

3.2.1.4 Read_FIFO

The GUSB_Platform_Read_FIFO function is for reading a response from the Controller.

Format:

```
int GUSB_Platform_Read_FIFO(unsigned int DeviceName,
                            unsigned int DeviceNumber,
                            t_USB_FIFO_Interface_Buffer *pRead,
                            unsigned int *DataLength)
```

Parameters:

DeviceName

Type of the addressed device (number declared in *GUSB_Platform_def.h*, for USB 3060/ basic MOST 3060 = 18)

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has <u>always</u> the DeviceNumber 1).

Pointer, for example pRead,
to the reading buffer
After successful execution of the function, there is the data in this reading buffer, consisting of Response Header and Response Bytes (currently max. 4096 bytes per response)

DataLength

Prior to function call: Size of the reading buffer in bytes (to be given)

After function execution: Number of bytes actually read

Description:

The GUSB_Platform_Read_FIFO function reads back the oldest response written by the Controller. In the case no response was received within the fixed Timeout of 100 ms, the function returns NO error, but the Number of bytes actually read is 0 !!!

3.2.1.5 Read_ FIFO_Timeout

The GUSB_Platform_Read_FIFO_Timeout function is for reading a response from the Controller within the Timeout to be given.

Format:

```
int GUSB_Platform_Read_FIFO_Timeout(unsigned int DeviceName,
                                    unsigned int DeviceNumber,
                                    t_USB_FIFO_Interface_Buffer *pRead,
                                    unsigned int *DataLength,
                                    unsigned int Timeout)
```

Parameters:

DeviceName

Type of the addressed device (number declared in *GUSB_Platform_def.h*, for USB 3060/ basic MOST 3060 = 18)

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has always the DeviceNumber 1).

Pointer, for example pRead,
to the reading buffer
After successful execution of the function, there is the data in this reading buffer, consisting of Response Header and Response Bytes (currently max. 4096 bytes per response)

DataLength

Prior to function call: Size of the reading buffer in bytes (to be given)

After function execution: Number of bytes actually read

Timeout

To be given in milliseconds (500 as a standard value)

Description:

The GUSB_Platform_Read_FIFO_Timeout function reads back the oldest response written by the Controller. In the case no response was received within the Timeout to be given, the function returns NO error, but the Number of bytes actually read is 0 !!!

3.2.1.6 Write_ COMMAND

With the GUSB_Platform_Write_COMMAND a configuration command is sent to the USB Controller.

Format:

```
int GUSB_Platform_Write_COMMAND(unsigned int DeviceName,
                                unsigned int DeviceNumber,
                                t_USB_COMMAND_Interface_Buffer *pWrite,
                                unsigned int DataLength)
```

Parameters:

DeviceName

Type of the addressed device (number declared in *GUSB_Platform_def.h*, for USB 3060/ basic MOST 3060 = 18)

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has <u>always</u> the DeviceNumber 1).

Pointer, for example pWrite,
to the write data area

DataLength

Size of the storage area pWrite is pointing to, in bytes
See also USB Controller Control Commands
(currently max. 64 bytes per command)

Description:

The GUSB_Platform_Write_COMMAND function sends a command to the USB Controller.

For the general structure, see the USB Controller Control Commands section.

3.2.1.7 *Read_ COMMAND*

The GUSB_Platform_Read_COMMAND function is for reading a response from the USB Controller.

Format:

```
int GUSB_Platform_Read_COMMAND(unsigned int DeviceName,
                               unsigned int DeviceNumber,
                               t_USB_COMMAND_Interface_Buffer *pRead,
                               unsigned int *DataLength)
```

Parameters:

DeviceName

Type of the addressed device (number declared in *GUSB_Platform_def.h*, for USB 3060/ basic MOST 3060 = 18)

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has always the DeviceNumber 1).

Pointer, for example pRead,
to the reading buffer
After successful execution of the function, there is the data in this reading buffer, consisting of Response Header and Response Bytes
See also USB Controller Control Commands
(currently min. 64 bytes per response)

DataLength

Prior to function call: Size of the reading buffer in bytes (to be given)

After function execution: Number of bytes actually read

Description:

The GUSB_Platform_Read_COMMAND function reads back the oldest response written by the USB Controller.

If several responses were provided by the USB Controller, up to two of these responses are written into the buffer of the USB Controller. More possibly provided responses get lost!

3.2.1.8 *Xilinx_ Version*

The GUSB_Platform_Xilinx_Version function allows reading out the version of the loaded XILINX firmware.

Format:

```
int GUSB_Platform_Xilinx_Version(unsigned int DeviceName,
                                 unsigned int DeviceNumber,
                                 unsigned int *Version)
```

Parameters:

DeviceName

Type of the addressed device (number declared in *GUSB_Platform_def.h*, for USB 3060/ basic MOST 3060 = 18)

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has <u>always</u> the DeviceNumber 1).

Version

XILINX software version

Description:

The GUSB_Platform_Xilinx_Version function can be used to read out the version number of the software loaded to the FPGA.

Example:

Version number **`2.34`** is returned as **`234`**, version **`2.60`** as **`260`**.

3.3 Programming with LabVIEW

3.3.1 LabVIEW via G-API

On the delivered CD there is a folder with VIs to call USB 3060/ basic MOST 3060 devices under LabVIEW.

The LabVIEW VIs use the functions of the GOEPEL G-API for this.

3.3.2 LLB using the Windows Device Driver

On the delivered CD there is a folder with VIs to call USB 3060/ basic MOST 3060 devices under LabVIEW.

The functions described in the Windows Device Driver section are used for this.

3.4 Further GOEPEL Software

PROGRESS, Program Generator and myCAR of GOEPEL electronic are comfortable programs for testing with GOEPEL hardware.

Please refer to the corresponding Software Manuals to get more information regarding these programs.

3.5 USB Controller Control Commands

The USB Controller is responsible for connecting the USB 3060/ basic MOST 3060 device to the PC via USB 2.0.

Messages (generally USB commands) required for configuration can be sent to this USB Controller.

3.5.1 USB Command Structure

A USB command consists of four bytes Header and the Data (but Data is NOT required for all USB commands!).

The header of a USB command has the following structure:

Byte number	Indication	Contents
0	StartByte	0x23 (“#” ASCII character)
1	Command	(0x..) used codes according to USB Commands
2	reserved	0x00
3	reserved	0x00

3.5.2 USB Response Structure

Same as a USB command, also the USB response consists of four bytes Header and the Data (but Data is NOT returned by all USB commands!).

The header of a USB response has the following structure:

Byte number	Indication	Contents
0	StartByte	0x24
1	Command	(0x..) used codes according to USB Commands
2	Length	Length depending on the command
3	ErrorCode	Returns the error code of the command

3.5.3 USB Commands

At present there is only the READ_SW_VERSION USB command available.

Command	Indication	Description
0x04	READ_SW_VERSION	Provides the firmware version of the USB Controller Response: Byte 4: low byte of generic software version Byte 5: high byte of generic software version Byte 6: low byte of software version of functional part Byte 7: high byte of software version of functional part